Scalable Cluster Administration - Chiba City I Approach and Lessons Learned

John-Paul Navarro, Rémy Evard, Dan Nurmi, Narayan Desai
Mathematics and Computer Science Division
Argonne National Laboratory
{navarro, evard, nurmi, desai}@mcs.anl.gov

Abstract

Systems administrators of large clusters often need to perform the same administrative task hundreds or thousands of times. Administrators have traditionally performed some time-consuming tasks, such as operating system installation, configuration, and maintenance, manually. By combining network services such as DHCP, TFTP, FTP, HTTP, and NFS with remote hardware control and scripted installation, configuration, and maintenance techniques, cluster administrators can automate these administrative tasks.

Scalable cluster administration addresses this challenge: What hardware and software design techniques can cluster builders use to automate cluster administration on very large clusters? We describe the approach used in the Mathematics and Computer Science Division of Argonne National Laboratory on Chiba City I, a 314-node Linux cluster; and we analyze the scalability, flexibility, performance and reliability benefits and limitations from that approach.

1. Introduction

In 1994 Thomas Sterling and Don Becker, then working for NASA, built the first commodity PC-based cluster from sixteen PCs [1][2][3]. This new platform quickly gained acceptance amongst computer and computational scientists for delivering super-computer performance for many applications at a fraction of super-computer cost. Today many educational and research organizations use the commodity PC-based cluster as their high-performance computing platform of choice.

While the PC-based cluster has brought high-performance computing to small and medium organizations, it has also become a significant presence amongst the largest super-computers in the world. See the Top 500 super-computers list at http://www.top500.org/ [4].

The trend towards larger clusters has introduced a new set of design and scalability challenges for cluster administrators [5].

To address these challenges administrators have looked for ways to automate administrative activities. The first step in automating cluster administration generally involves setting up one or more hosts running network services used in automated boot, build, monitor and other cluster administration activities. Common network based administration support services include:

- DHCP for network initialization
- TFTP for boot image delivery
- FTP, HTTP, NFS, and SMB for file delivery
- A configuration information delivery service
- SNMP for monitoring and event notification
- SYSLOG for OS and core service logging
- DNS for host name resolution
- NTP for time synchronization

We call machines providing these services "management servers". The challenge is to determine how many management servers are needed and how the services are distributed between them.

Most clusters require very few management servers. For all small clusters, and most medium to large clusters, an organization's existing infrastructure servers can provide these services to clusters.

To determine whether one management server is sufficient an administrator must determine how many cluster clients will concurrently use the services and how quickly these management service requests must be handled.

In addition, services used during different phases in a machine's life cycle may have totally different usage patterns and performance characteristics. For example, network interface configurations with DHCP and boot image delivery with TFTP are generally only used at machine boot time and require relatively little management server capacity to service. File delivery services using protocols such as FTP, HTTP, NFS, and SMB are used heavily during machine build or upgrade. These services also demand significant storage and network capacity to house and deliver the files. Logging (SYSLOG), monitoring (SNMP), and remote console services are most important during normal operations and typically have constant low to moderate requirements.

As the size of a cluster grows and the build and reconfiguration rates increase, the demand for these services may exceed the capabilities of a single machine or of the available infrastructure servers. Two options to address this management scalability limitation include separating services onto different machines and upgrading servers so they can service more requests. These steps are generally enough to scale most management services for large clusters.

What are the scalability characteristics of the various services? Some services, such as file delivery services, may need to deliver hundreds of megabytes in order to build a single node. How does one scale these services so that hundreds of nodes can rebuild in a short period of time? This paper presents the scalable cluster management approach used to address these and similar questions on Chiba City I at Argonne National Laboratory.

2. Importance of Management Scalability

For most clusters, management scalability is not an issue, because the number of managed nodes is low, the rebuild and reconfigure rates are low, or the rebuild and reconfigure performance is not a concern.

Management scalability is important on Chiba City because its primary purpose is to be a *scalability testbed*, built from *open source* components, for the high-performance computing and computer science communities. As a scalability testbed, Chiba City is dedicated to the research, development, and testing of architectures, algorithms, software, and protocols that push the scalability boundary of clusters and the applications that run on them.

Although built and operated primarily using open source software, the goal is for Chiba City software to support installation and operation of any open or closed-source operating system on non-management nodes. In support of this objective we developed a cluster administration toolkit, called the City Toolkit [7], designed to support the unattended installation of arbitrary operating systems. Because Chiba City is a testbed we need to rebuild and reconfigure nodes frequently. The combination of a large node count and a high rebuild and reconfiguration rate makes a scalable cluster management design critical.

With a scalable cluster management design, rebuilds, reconfigurations, and general management activities can run quickly, efficiently and will improve overall cluster availability and reliability.

Using the City Toolkit, we can change a master configuration database that specifies the desired image (OS + configuration) for any number of managed nodes and can complete a parallel rebuild in less than 30 minutes. Our long-term goal is to support node rebuilds on demand; for example, to support dynamic OS requirements on a per job basis and so that a kernel developer is able to quickly rebuild 500 nodes with a custom kernel for scalability testing and debugging.

Another reason scalable management is important is that with the widespread adoption of clusters for high-performance computing, we believe the research and commercial communities will deploy clusters with tens and possibly hundreds of thousands of nodes in the next 20 years. As a scalability testbed, Chiba City must be able to provide an environment where researchers can investigate scalable software and management challenges that will undoubtedly affect all very large commodity PC-based clusters.

3. Chiba City Management Architecture

Figure 1 shows all the components in Chiba City.

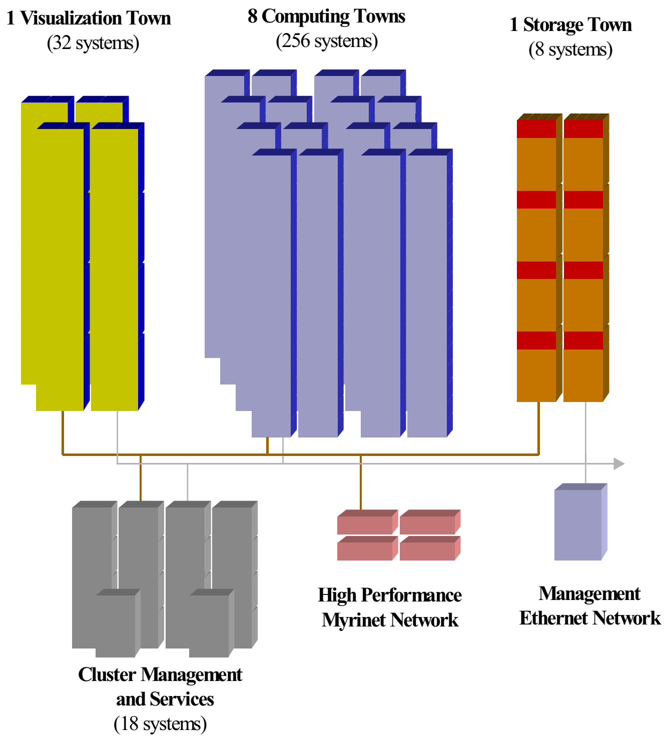

Fig. 1. Chiba City component diagram

Physically, Chiba City comprises blocks of physically contiguous nodes called towns. A town consists of between 8 and 32 application-usable nodes, one management node called a mayor, an Ethernet network switch, serial concentrators connecting node serial ports to the mayor, and network-addressable remote power controllers. Most Chiba City towns fit into two racks. The only management-related cables coming from the two town racks are dual Gigabit Ethernet uplinks, one management Ethernet network uplink, the mayor's serial port cable, and five power cables. A town has all the hardware necessary to operate as an independent subcluster. As a matter of fact we've taken one such Chiba City town to a conference.

Towns are not merely a physical partitioning scheme. The concept defines how management software and services--required to build, configure, and operate the nodes in a town--work. An example of that physical and management link is the boot-and-build process for nodes. When a node boots, it sends its boot loader prompt (LILO or GRUB) over its serial port. The town mayor, which runs serial port monitoring software, detects that boot loader prompt and sends a response, telling the node either to boot or build. The build tools used to build a node and the software packages installed on that node are accessed from the mayor via NFS or HTTP/FTP.

The mayor provides the following services:

- Relay or proxy access to the master cluster database
- DHCP for configuring network interfaces
- TFTP access to boot images
- Console monitoring, logging, and interactive access
- Root NFS used during node build and debugging
- NFS access to non-local software
- NFS, HTTP, or FTP access to software packages installed during a build
- NFS-based file relay service for copying files to or from the nodes before and after job runs (Chiba City doesn't have a global general-purpose file system)
- Relay service for global management commands

Managing the town mayors, login machines, file servers, and other servers is a master management server called the president. The president behaves like a mayor to the nodes it manages. In addition, the president contains the master copy of the images (OS + configuration), layered software packages, and the master cluster configuration database. To synchronize software between the president and the mayors we use rsync [10]. Access to the master configuration database on the president is available through proxies on mayors.

We chose this three level hierarchical design because we felt it could architecturally be extended to more levels, thus providing a form of scalability. Using a ratio of 32 managed nodes to one management server we could build a 1024-node cluster with the current three levels in Chiba City. Using the same managed to management node ratio, using four hierarchical levels we could build a cluster with over 32,000 nodes.

As an example of how this management hierarchy affects administrative activities, the following steps describe the process of modifying a node image and rebuilding nodes with that image. Images, as we defined them in the City tools, are the combination of disk initialization information (partitions, file-system types, and initial file-system contents), a collection of scripts that drive the build process, and a set of software packages and configuration files.

1. Change the node image on the president (for example, change the desired root file-system size).
2. Rsync all the images from the president to the mayors.
3. Modify the master configuration database specifying which nodes need the new image.
4. Reboot or remotely power cycle the nodes that need the new image.

The nodes then boot and are directed by their mayor to build with the database specified image. When the nodes finish building they update the master configuration database.

4. Elements of the Chiba City Management Architecture That Scaled Well

Overall we found many aspects of the Chiba management architecture worked well on a 314-node cluster. The following sections describe these aspects that, in principle, we feel could be used to manage clusters with tens of thousands of nodes.

4.1 Dedicated Management Servers

An important aspect of our design is that management servers are not available to user applications. Although that design decision seems obvious to us, many people consider this a hardware tax because the hardware isn't available to cluster users. Separating management servers is particularly important in our environment because most cluster applications expect dedicated node access. Dedicated node access is important for many parallel applications because they execute on multiple nodes in lock-step and when any single node experiences slowdown all the nodes running that application are affected.

Our cluster build-and-configure software was designed to run in a RedHat Linux environment. The software requirements for the management services are often different from those of the application software. For example, currently our management and software runs under RedHat 7.1, but is capable of installing on user nodes other versions of RedHat, different distributions such as Mandrake, or other operating systems such as FreeBSD.

4.2 Specialized Management Server Hardware

Management servers need a hardware configuration tuned for running management services. This hardware configuration may often be different from what user application require. For example, management servers may require hundreds of gigabytes of disk to store or cache software packages, high-performance gigabit commodity networks to handle high conventional TCP/IP communication loads, and multiple CPUs to handle very high context switching rates associated with many clients making simultaneous management service requests. To fulfill these hardware requirements, high-end large footprint machines are needed. In contrast, large clusters typically look for very small footprint machines for workhorses nodes used by cluster applications. One thousand 1U application nodes can require 25 standard racks. If a compute node is 2U, 3U, or 4U, the necessary floor space doubles, triples, or quadruples.

Selecting the appropriate management server hardware is more important than we had originally thought. Care should be taken that the appropriate amount of RAM, number of CPUs, disk space, I/O bandwidth, and network bandwidth are available on management servers.

4.3 Master Management Server

Once settled on dedicated management servers, one needs to decide how to distribute services between those servers. The approach used on Chiba City was to give a single machine the role of master management server, which we called the president. A president is the authoritative source for all software and configuration information, the recipient of all status information, and the point from which all administrative function could be issued. We found this model to be straightforward and easy to use. From the president we have the ability to build, configure, and update other management servers and to initiate management operations that are forwarded to the mayors responsible for the desired target nodes. We believe that a canonical source for all software, from which all management commands might be issued, should scale to any size cluster. For this to be the case, though, management software must divide and delegate operations to subordinate management servers.

4.4 Rebuildable Management Nodes

Once we established that we had single master management server, the president, we developed procedures for rebuilding other management servers. It then became possible to rebuild a management node and transfer the president role to it using only a few manual steps, in effect upgrading the president. The scalability advantage in automating management server builds is that it’s easy to add more management servers as the number of managed nodes grows.

4.5 Remote Power Control

Remote network-based power control is an essential component of hands-off administration. Without it, some administrative and operational activities would require individuals walking up and down aisles of computer racks pushing buttons. This is impractical, not just from a human time/cost perspective, but also because of the likelihood of pushing the wrong button.

4.6 Remote Console

We found a remote console to be a useful tool for node identification, boot control, and network failure diagnosis and recovery. In vary large clusters a remote console may not be essential if network or other techniques are used to identify nodes and control the boot process.

4.7 Parallel Management Algorithms

Early in the City Toolkit design we realized that along with parallel hardware and management servers independently managing a subset of nodes, we also needed parallel management software algorithms. One very effective tool we used heavily is PDSH [8]. PDSH provides a simple thread based model for executing commands on multiple nodes in parallel.

As mentioned above, another essential design point in scalable management software is delegation. By taking a single centrally issued command, splitting it into parallel components, and delegating those to other management servers, arbitrary management operations can scale to thousands of nodes and can execute quickly.

5. Elements of the Chiba Management Architecture That Did Not Scale Well

Two and a half years of operational experience with the current Chiba City management architecture has revealed several deficiencies that we feel must be addressed in order to effectively administer even larger clusters. These deficiencies can be grouped into the following categories: limited flexibility, poor performance, high overhead or complexity, and lack of fault tolerance. Key deficiencies follow.

5.1 Hard Mapping between Managed and Management Nodes

The most unscalable and problematic aspect of the Chiba City management architecture was the direct dependence between a fixed set of contiguous nodes and a single management server. One consequence was very poor management service load distribution where a subset of management servers would be overloaded while the remaining servers were effectively idle.

5.2 Management Node Failures

When a mayor fails, all of the nodes beneath it in the hierarchy also fail due to NFS, DHCP, console, and other dependencies. Moreover, it is also impossible to rebuild or reconfigure dependent nodes because the client-to-mayor link is wired in hardware and software designs.

In the worst-case scenario, where the president fails, the entire cluster becomes unusable because most administrative activities depend in some way on the central configuration database.

5.3 Console Access through Management Nodes

Having console access tied to specific mayors meant that if a mayor was unavailable all the nodes it managed would be unusable.

5.4 All Management Services on Every Mayor

To make every town operate as an independent sub-cluster we configured every mayor with all the management services. This is very inefficient and a major headache to support since many services could be provided by fewer servers, or even one. Two examples are DHCP and the master configuration database. In our particular case in which all Chiba City was under a single flat IP space, running 11 DHCP servers on the same subnet was a challenge.

5.5 Unsuitability of Some Services for a Hierarchy

Some management services, such as DHCP, NTP, TFTP, and console are more difficult to install and operate hierarchically (and don't require or benefit from it).

5.6 Ratio of Management Server to Managed Node

It's impossible to pick a single "right" ratio of mayor to managed nodes for all services because each service has different scaling characteristics. For example, the number of concurrent NTP or DHCP requests that a single server can handle is very large, probably in the hundreds or thousands, while the number of concurrent lHTTP or FTP large file get requests that could be handled is quite small (in the 10-20 range if the management node has approximate 10x better Ethernet bandwidth than the nodes it manages). If both of these classes of service are configured in a fixed 32 client to 1 server ratio, as was the case on Chiba City, we end up having more NTP and DHCP servers than are needed and fewer HTTP and FTP servers than could be effectively utilized.

Consequently we found a 32 to 1 ratio compromise misses almost all the optimal client to server ratio targets.

5.7 Hierarchical Configuration and Software Push

Pushing configuration changes and commands down the three level hierarchy is a time consuming process. Having to deal with four or five levels would be a serious problem. For example, to apply software or configuration changes to nodes requires applying the change on the president, pushing the change iteratively down the hierarchy, and applying the change on the target machines. This multi-step, fixed-path design is vulnerable to cascading failures. At one point we had 200 MB of software required on every management node; pushing this much software every time we changed something was a slow and painful process, even when using a smart push tool like rsync.

5.8 Configuration database bottleneck

Most administrative cluster activities require information from the configuration databases. Some notable examples include information on: whether to boot or rebuild a machine when it powers up, which image, software packages, and configurations to apply to a node being rebuilt, and which users have access to which nodes.

The combined effect of this direct database access is that our database could be overwhelmed during intense build, configuration, or job start/exit activity.

6. Conclusion

Management scalability is a hard problem. It's also a moving target. As management scalability problems are identified and fixed we are able to build and operate larger clusters. With larger cluster we find new scalability challenges that were not apparent at a smaller scale.

Our first architecture has taught us many valuable lessons and Chiba City I has proven to be an excellent platform to carry out scalable design experiments. This has been the case because we have the ability to take interesting new ideas and deploy them to establish real benefits and disadvantages.

In the category of flexibility we have learned that:

- The ratio of managed node to management server needs to be based on the scalability characteristics of each service instead of a hard ratio derived from the scaling characteristics of the least scalable service. A design is needed that will allow use of different ratios for different services.
- In clusters with serial console, tying node consoles to a single management server limits where the services using that console may run. We need a more network-like model for console access that makes consoles available to many services on multiple management servers.
- While a hierarchical approach may work well for some services it doesn't work well for others. We need a more flexible design that can distribute services between as many machines as necessary using a topology appropriate for each service.

In the category of performance we've learned that:

- Statically linking managed node to a specific management node leads to poor server load distribution. This directly affects management service performance.
- Instead of using multiple steps to push software down the hierarchy we need more intelligent demand based caching. If software and configuration changes are cached on demand, applying configuration changes could be more automated and efficient.
- NFS performs poorly.

In the category of Fault Tolerance we have learned:

- By depending on a single president for most administrative activities we have created a single point of failure that can bring an entire cluster down.
- When managed nodes depend on specific management nodes any management node failure can bring parts of a cluster down. Those part of the cluster will remain down until the failed management node is fixed.
- NFS failure situations are difficult to recover from. We need to use more fault tolerant networking protocols between managed and management nodes.
- The net effect of making managed nodes depend on individual management nodes, or a single president, is that any management failure gets amplified to the remaining cluster nodes. More research is needed to reverse this amplification effect.

7. Future Work

Based on our experience we have identified several management design changes that we intend to investigate which could lead to architectures capable of managing thousands of nodes.

To address the single point of failure problem with the president while maintaining the advantages of an authoritative configuration, software, and administrative machine, we are going to explore applying high-availability techniques to the president. One of the leading president services that could benefit from high-availability is the master configuration database.

More research into stateless network protocols and dynamic or flexible managed to management node links could lead to a solution to the management node failure amplification effect and to the unbalanced management server utilization problem.

Although the hierarchy has in some ways been useful, we believe research needs to be done into alternate topologies tuned to the specific requirements of various management services.

Another intriguing management service design approach that we believe needs to be explored is using parallel application programming techniques, such as MPI, to design and develop these services. We see parallelization of management applications and commands as a promising scaling technique. There is no reason to believe that the scalability benefits achieved by applications using parallel techniques could not apply to management services and applications.

Finally, to avoid pushing complete software repositories between management servers, additional research should be done into using demand based software distribution and caching techniques.

8. Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

9. References

[1] www.beowulf.org Information on Beowulf and Commodity PC based clusters.

[2] D. J. Becker, T. L. Sterling, D. F. Savarese, J.E. Dorband, U.A. Ranawak, and C.V. Packer. Beowulf: A Parallel Workstation for Scientific Computation. *Proceedings of the International Conference on Parallel Processing*, 1995.

[3] T. L. Sterling, J. Salmon, and D. J. Becker. How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters. MIT Press, 1999.

[4] www.top500.org. Top 500 super-computers in the world.

[5] D. S. Greenberg, R. B. Brightwell, L. A. Fisk, A. B. Maccabe, and R. E. Riesen. A System Software Architecture for High-End Computing. *Proceedings of Supercomputing '97*, 1997.

[6] J. Challenger, P. Dantzin, and A. Iyengar. A Scalable and Highly Available System for Serving Dynamic Data at Frequently Accessed Web Sites. *Proceedings of Supercomputing '98, 1998.*

[8] The City Toolkit: http://www.mcs.anl.gov/systems/software/

[8] The parallel distributed shell (PDSH): http://www.llnl.gov/icc/lc/pdsh.html

[9] Chiba City Design: http://www.mcs.anl.gov/chiba/

[10] File synchronization using rsync. http://www.samba.org/rsync.